Simple Usage of gRPC with Python

Barkhayot Follow 2 min read · Dec 29, 2023

106 1

Introduction

gRPC (gRPC Remote Procedure Calls) is an open-source framework developed by Google that facilitates communication between distributed systems by defining and implementing efficient and extensible RPC (Remote Procedure Call) APIs. In this blog post, we'll explore a simple example of using gRPC with Python to create a basic client-server application.

Setting Up

Before we dive into the code, make sure you have the necessary tools installed

```
pip install grpcio grpcio-tools
```

Defining the Service

The first step is to define the service using Protocol Buffers (protobuf). Create a file named `calculator.proto` with the following content:

```
syntax = "proto3";

package calculator;

service Calculator {
  rpc Add (AddRequest) returns (AddResponse);
}

message AddRequest {
  int32 num1 = 1;
  int32 num2 = 2;
}

message AddResponse {
  int32 result = 1;
}
```

Generating gRPC Files

Run the following commands to generate the gRPC files:

```
python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. calculator.pr
```

This will generate `calculator_pb2.py` and `calculator_pb2_grpc.py`.

Implementing the Server

Create a file named `server.py`:

```
import grpc
from concurrent import futures
import calculator_pb2
import calculator_pb2_grpc

class CalculatorServicer(calculator_pb2_grpc.CalculatorServicer):
    def Add(self, request, context):
        result = request.num1 + request.num2
        return calculator_pb2.AddResponse(result=result)

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    calculator_pb2_grpc.add_CalculatorServicer_to_server(CalculatorServicer(), s
    server.add_insecure_port('[::]:50051')
    server.start()
    server.wait_for_termination()

if __name__ == '__main__':
    serve()
```

Implementing the Client

Create a file named `client.py`:

```
import grpc
import calculator_pb2
import calculator_pb2_grpc

def run(num1, num2):
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = calculator_pb2_grpc.CalculatorStub(channel)
        response = stub.Add(calculator_pb2.AddRequest(num1=num1, num2=num2))
    print("Result: " + response.result)

if __name__ == '__main__':
    # Get user input
    num1 = int(input("Please input num1: "))
    num2 = int(input("Please input num2: "))
    run(num1, num2)
```

Running the Server and Client

In one terminal, run the server:

```
python server.py
```

In another terminal, run the client:

```
python client.py
```

You should see the result of adding 10 and 20 printed by the client.

Conclusion

This example demonstrates a basic usage of gRPC with Python for building a simple client-server application. gRPC provides a powerful and efficient way to define and implement APIs for communication between distributed systems. Explore further to discover more features and use cases for gRPC in your projects.

Feel free to experiment with additional gRPC features, error handling, and security options to enhance your understanding of this versatile framework. The official gRPC documentation is an excellent resource for diving deeper into advanced topics and best practices.

Source code : https://github.com/barkhayot/grpc-python-example

Happy coding!

Grpc Python Backend Microservices Distributed Systems

106 1

Written by Barkhayot
7 Followers · 2 Following
Follow

Responses (1)

Write a response

What are your thoughts?

Abraham Ayemigah
Oct 31, 2024

Good content but it will be great explaining some parts of your codes.

Reply

More from Barkhayot

Barkhayot

Building a Flask API Gateway for gRPC Microservices: A Practical...

In this post, we'll explore the process of creating a simple Flask API Gateway to...

Jan 10, 2024 3

Barkhayot

How to simply detect and fix memory leaks in Go with pprof?

Memory leaks can be challenging, even in languages like Go that have automatic...

Nov 2, 2024 20

Barkhayot

Optimizing ClickHouse Connections in Go

ClickHouse is a powerful columnar database management system well-suited for real-tim...

Jun 28, 2024 20

Barkhayot

Demystifying gRPC: A Comprehensive Guide to High-...

In the realm of modern software development, the quest for efficient and...

Dec 27, 2023 17

See all from Barkhayot

Recommended from Medium

Mikail Fatmy

Protobuf, JSON, and XML: A Practical Comparison

Data serialization is at the heart of modern software development. Whether you're...

Dec 2, 2024 9

In Mindful Engineering by Rahul Suthar

Understanding Message Brokers and Message Backends: Redis,...

Now, let's explore Kafka, a powerful message broker used for high-throughput, low-latenc...

Dec 17, 2024 33 1

In TDCH by Py-Core Python Programming

Using APIs with Python: SOAP vs REST vs GraphQL

In the world of software development, APIs (Application Programming Interfaces) are...

3d ago 61

Debugged Thoughts

Why Big Tech Is Quietly Abandoning Golang

In the early 2010s, Go (Golang) was hailed as the future of systems programming. It was...

Apr 21 332 27

Abhinav

Zig Is Quietly Doing What Go Fans Always Wanted

Go, also known as Golang, has been a darling of the programming world since its release...

6d ago 63

In OpenSourceBytes by C. L. Beard

13 Self-hosting Tools to Replace Paid Tools

Leave SaaS lock-in behind with these lightweight, self-hosted alternatives

1d ago 168 5

See more recommendations